Using Supervised Pretraining to Improve Generalization of Neural Networks on Binary Classification Problems

Alex Yuxuan Peng[1], Yun Sing Koh[1], Patricia Riddle[1], and Bernhard Pfahringer[2]

[1] University of Auckland, Auckland, New Zealand
ypen260@aucklanduni.ac.nz, {ykoh, pat}@cs.auckland.ac.nz
[2] University of Waikato, Hamilton, New Zealand
bernhard@cs.waikato.ac.nz

Abstract. Neural networks are known to be very sensitive to the initial weights. There has been a lot of research on initialization that aims to stabilize the training process. However, very little research has studied the relationship between initialization and generalization. We demonstrate that poorly initialized model will lead to lower test accuracy. We propose a supervised pretraining technique that helps improve generalization on binary classification problems. The experimental results on four UCI datasets show that the proposed pretraining leads to higher test accuracy compared to the he_normal initialization when the training set is small. In further experiments on synthetic data, the improvement on test accuracy using the proposed pretraining reaches more than 30% when the data has high dimensionality and noisy features.

Keywords: Neural network · Pretraining · Initialization · Generalization.

1 Introduction

Neural networks have attracted a lot of attention from both academia and industry due to their success in different machine learning tasks. Recurrent neural networks (RNNs) have been successfully applied to speech recognition and natural language translation [8, 17, 19]. Convolutional neural networks (CNNs) have been the winning models for many image understanding challenges [9, 12, 14]. Fully-connected neural networks (FNNs) were shown to be very effective at identifying exotic particles without features hand-engineered by physicists [2].

Apart from novel network architectures, the success of neural networks in so many applications is a result of many advancements in the basic components such as activation functions and initialization. During the training process deep neural networks are sensitive to the initial weights. A poor choice of initial weights can result in very slow training, "vanishing gradient", "dead neurons" or even numerical problems. A few initialization methods have been proposed to

stabilize the training process [7, 11, 15]. However, they do not aim to improve the generalization of neural networks. It is possible that even though these methods can successfully improve the stability and speed of training, they are not necessarily the best for improving the generalization of neural networks. Initialization of neural networks is a very difficult problem to study due to the complex nature of the cost surface and the optimization dynamics. However, more research needs to be devoted to exploring the connection between the initialization and the generalization of neural networks. We argue that the initialization affects the generalization of neural networks. Intuitively speaking, the initial weights of a neural network can be thought of as the prior of a model. If this intuition is correct, the choice of initial weights will have a big effect on which model we get after training, when there is a lack of labelled training data. Hence, studying the effect of initialization on generalization is very valuable to domains where labelled data is not easy and cheap to collect.

The **main contribution** of this paper is a supervised pretraining that improves the generalization of fully-connected neural networks (FNNs) on binary classification problems. We firstly demonstrate that poorly initialized models lead to poorer generalization. We then propose a supervised pretraining method that improves the generalization when labelled data is limited, by taking advantage of unlabelled data. The proposed method trains a neural network to distinguish real data from shuffled data. The learned weights are reused as initial weights when training on the labelled training data. The intuition is that during the pretraining, the model has to learn joint distributions of the real data in order to identify real data points from shuffled ones. And the learned weights might be a better prior than standard initialization methods that sample values from normal or uniform distributions. The improvement in generalization by using the proposed method is shown to be more obvious when there is a lack of labelled data and the class separation is small. Experimental results also suggest that the proposed pretraining works best when the data has high dimensionality and contains noisy features. We only consider binary classification problems in this work. We measure generalization using test accuracy in all our experiments.

The rest of the paper is organized as follows. In Section 2, we review related works. In Section 3, we demonstrate that initialization of neural networks affects generalization. In Section 4, we propose a supervised pretraining method to improve the generalization of FNNs on binary classification problems. We conducted experiments on both UCI datasets and synthetic datasets to evaluate the proposed method in Section 5. In Section 6, we discuss some issues with the proposed pretraining method. Finally, we conclude our work and propose future works in Section 7.

2 Related works

Glorot and Bengio [7] derived a way to initialize weights depending on the number of input units and output units of a layer. They derived this method by assuming only linear activation functions are used, and attempting to initialize

the weights in such a way that the variances of activation values and gradient across all layers are the same at the beginning of training. Despite the unrealistic assumption of a linear activation function, this initialization works well in practice. Using the same idea, He et al. [11] derived an initialization method specifically for the ReLU activation function depending on the number of input units of a layer. It draws weight values from a normal distribution with mean value of 0 and the standard deviation $sqrt(2/fan\_in)$, where fan_in is the number of input units of a layer. We call this initialization method *he_normal*. They showed that he_normal performed well even for really deep networks using ReLU while the Glorot and Bengio method [7] failed.

Saxe et al. [18] recommended initializing weights to random orthogonal matrices. However one needs to carefully choose the gain factor when a nonlinear activation function is used. Mishkin and Matas [15] proposed a method called layer-sequential unit-variance (LSUV) initialization. LSUV firstly initializes the weights with orthonormal matrices, and then normalizes the activation values of each layer to have variance of 1 through a few forward passes.

Unsupervised methods such as restricted Boltzmann machines and autoencoders were used as layer-wise pretraining to initialize deep neural networks [3, 13, 16]. This was done to solve the "vanishing gradient" problem in training deep neural networks, before piece-wise linear activation functions were widely adopted. However, it was later discovered that unsupervised pretraining actually improves generalization when labelled training data is limited [6]. When labelled data for a task is limited, supervised pretraining on a different but related task was found to improve the generalization on the original task [20].

3 Demonstration that initialization affects generalization

Most machine learning problems can be reduced to search problems, especially when an iterative learning algorithm is used to find the final model. A search problem usually consists of a search space, an initial state, a target state and a search algorithm. Figure 1 describes training neural networks as a search process. The area inside the ellipse represents the representation capacity of a neural network. The larger the neural network, the bigger the representation capacity and the larger the search space. The square represents the initial state of the model and the triangle is the final state of the model. The circular dots are the model states the learning algorithm has visited. The model search space can be discrete or continuous. But in neural networks, the search space is usually continuous. And the search space or representation capacity of a neural network is not completely independent of the search process. The bigger and more complex a neural network is, the more difficult it is to train. So the representation capacity can potentially affect where the the model will end up while holding the initial state and learning algorithm constant.

Unlike many search problems where the target state is known, we usually do not know the target state in machine learning. And in the case of neural networks, we cannot analytically find the optimal model state due to the non-

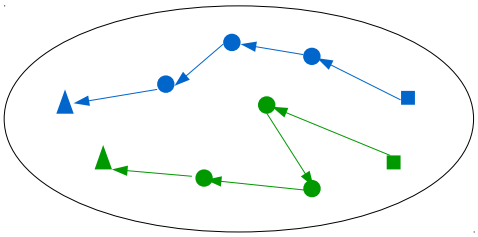
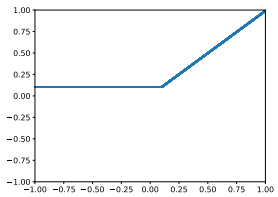


Fig. 1. Machine learning represented as a search process. The area inside the elliptical boundary is the representation capacity of the model chosen. The circular dots are different model (parameter) states a learning algorithm has visited. The square is the initial state of the model while the triangle is the final state of the model.

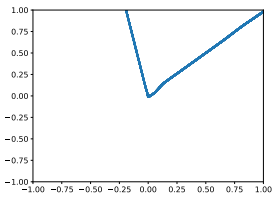
convex loss function. So iterative optimization algorithms are used in training neural networks. The most common learning algorithms in training neural networks are stochastic gradient descent and its variants. The first order derivatives of the model parameters are used as a heuristic for determining the next step. Obviously, different learning algorithms can result in different final models even if both the search space and initial state are kept the same.

Another factor that affects the final state of the search space is the initial state of the model. Different initial weights can lead a neural network to converging to different states, as illustrated in Figure 1. We argue that initialization affects the generalization of neural networks, especially when the training data is limited. The initial weights of a neural network can be seen as the prior of the model. The choice of the initial weights does not affect the generalization much if there is plenty of training data, as long as the initial weights do not lead to training problems. However, when there is a lack of training data, the prior will have a larger effect on the final state of the model. Figure 2 shows the plots of four fully connected networks trained to fit the same two data points but with different initial weights and activation functions. The two data points were (0.1, 0.1) and (0.5, 0.5). All four of the models had 128 units in each of the first two layers and one output unit. The initial weights were drawn from truncated normal distribution with mean of 0 and different standard deviation values (0.004 and 0.4). Any value that was more than two standard deviations away from the mean was discarded. All biases were set to 0. We used both rectified linear unit (ReLU) and hyperbolic tangent (tanh) as activation functions. We trained the model until the training loss became 0 and plotted the model. It can be seen that when the standard deviation of the initial weights is small, the learned function is relatively simple. When the standard deviation of the initial weights is big, the final model is much more "complex" and less likely to generalize well on unseen data.

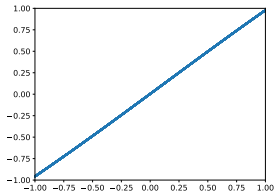
Similar experiments cannot be applied to real datasets and larger networks, because initial weights drawn from a normal distribution with a large standard



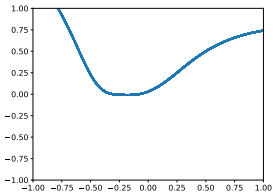
(a) Mean 0 and sd 0.004 (ReLU)



(b) Mean 0 and sd 0.4 (ReLU)



(c) Mean 0 and sd 0.004 (tanh)



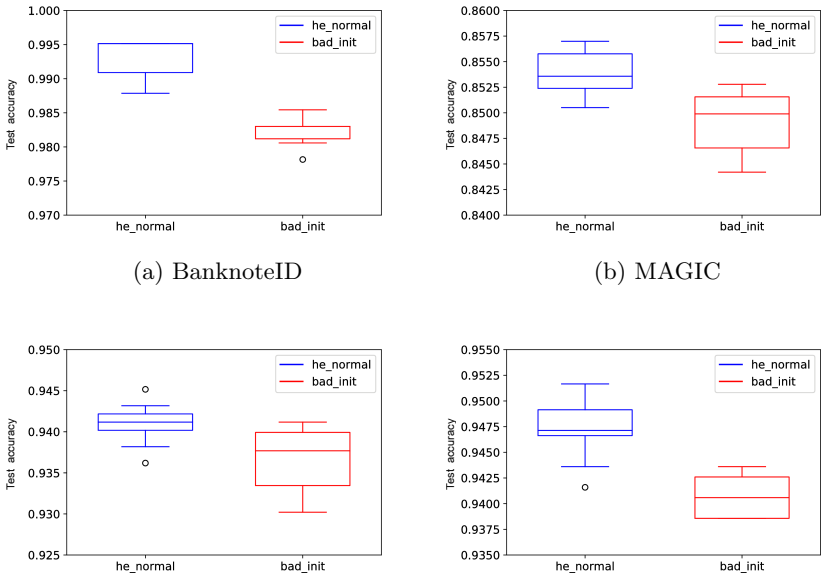
(d) Mean 0 and sd 0.4 (tanh)

Fig. 2. Plots of a three-layer neural network trained on two data points (0.1, 0.1) and (0.5, 0.5). The activation functions used are ReLU and tanh. The initial weights were drawn from normal distributions with different mean and standard deviation values. When the initial weights have a small standard deviation, the learned model is more linear ("simpler"). The learned model becomes more complex and less likely to fit unseen data well, when the initial weights have a higher standard deviation.

deviation leads to various training problems such as "dead neurons" and exploded activation values. Inspired by transfer learning [20], we decided to learn inappropriate initial weights for the original classification task by pretraining a model on randomly shuffled data. More specifically, we shuffled the attribute values of the original data and randomly attached labels to the shuffled data according to the uniform distribution. We obtained a set of weights by training a model on this shuffled data. We expected this set of weights to be a very poor prior for the original classification problem and thus would lead to lower test accuracy. When training a model on the shuffled data, we used he_normal as the initialization method. These learned weights were then used as initial weights when training on the original data. We then compared the model initialized with these transferred weights against the model initialized using he_normal [11].

We carried out the experiments on four different datasets: BanknoteID, MAGIC, Waveform and Waveform_noise. Details of these datasets can be found in Section 5.1. We used a five-layer fully connected network for all the experiments. Each layer has 1024 units except the last layer. The last layer is a softmax layer. The activation function used was ReLU. We used the full-batch gradient descent with a constant learning rate of 0.01. We chose the number of epochs to make sure both models had converged. The number of epochs we used for

each dataset can be found in the second column of Table 3. The training on the shuffled data was run 10000 epochs for all the cases. We ran each experiment 10 times. Figure 3 shows the distribution of test accuracies obtained using the transferred initial weights and he_normal on four datasets. As can be seen, in all four cases, the model initialized with the transferred weights performed worse.



(a) BanknoteID (b) MAGIC

(c) Waveform (d) Waveform_noise

Fig. 3. Comparison of the model initialized with transferred weights (red) against the model initialized with he_normal (blue) in terms of test accuracy. We ran each case 10 times.

In this section, we demonstrated that a poorly initialized model does indeed lead to lower test accuracy. In the next section, we propose a supervised pretraining method that improves generalization, especially when labelled training data is limited.

4 Supervised pretraining

We propose a supervised pretraining method that takes advantage of unlabelled data to improve the generalization of the original classification problem. The underlying assumption of this method is that the weights learned during the supervised pretraining are a good prior for the original classification problem.

This can happen if the pretrained model contains information about the joint distributions of the attributes in the original data.

The pretraining phase is basically a binary classification that attempts to identify real data from shuffled data. Shuffled data does not provide us with real patterns but instead are considered as noise. The learned weights are then reused in the original classification problem.

Table 1. An example of how the pretraining data is generated. Table (a) is the original unlabelled data and Table (b) is the labelled training data. Table (c) is the shuffled unlabelled data. Note that no data point in the shuffled data is from the unlabelled data. Table (d) is the pretraining data. The pretraining data is created by stacking the original unlabelled data and shuffled data, and labelling them as 1 and 0 respectively.

Attr 1	Attr 2	Attr 3
1	1	1
2	2	2
3	3	3

(a) unlabelled data

Attr 1	Attr 2	Attr 3	Label
4	4	4	1
5	5	5	0

(b) Labelled data

Attr 1	Attr 2	Attr 3
1	2	3
2	3	1
3	1	2

(c) Shuffled data

Attr 1	Attr 2	Attr 3	Label
1	1	1	1
2	2	2	1
3	3	3	1
1	2	3	0
2	3	1	0
3	1	2	0

(d) Pretraining data

We discuss the supervised pretraining method in detail here.

1. We start by randomly shuffling the attribute values of the unlabelled data across rows. Each attribute is shuffled independently from other attributes. This is to break the joint distributions of the original data, but to keep the distributions of individual attributes unchanged. Table 1(a) shows an example of unlabelled data and Table 1(c) shows an example of shuffled data. Note that the shuffling is random, so there can be many different examples of shuffled data generated from the same unlabelled data. In our implementation, we ensured that there was no original real data points in the shuffled data due to chance collision. Any data points in the shuffled data that could be found in the original data were replaced by new shuffled data points. This was done to avoid confusing the model during the pretraining process. Note that labelled training data can be added to the unlabelled data after taking out the labels, in order to create a larger set of unlabelled data. A shuffled dataset can then be created from this enlarged unlabelled dataset.

2. The unlabelled data and shuffled data are stacked together. Then all the real data points are assigned one label and all the shuffled data points are assigned a different label. This is the pretraining data used in our method. The specific values being used as labels do not matter, as long as all the real data points are assigned the same label and all the shuffled data points are assigned a different label. This is because one hot encoding is used to encode the labels in our models. In our implementation, the real data is labelled as 1 and the shuffled data is labelled as 0. The pretraining data always has a balanced class distribution. Table 1(d) shows an example of pretraining data.
3. The pretraining data is then split into a training set and a validation set using stratified sampling. In our implementation, 70% of the pretraining data was used as training data and the rest was used as a validation set.
4. During the pretraining, a neural network model is trained on the pretraining data. The validation set is used to stop the pretraining early. For the pretraining, the model was initialized using he_normal in our implementation.
5. Finally, the weights learned in the pretraining are reused as initial weights when training a model on the labelled training data shown in Table 1(b). This is sometimes called transfer learning, where weights learned in one problem are transferred to another problem [20]. In our implementation, all the layers except the last one were transferred from the pretrained model. The last layer was replaced with weights initialized using he_normal. We did not freeze any layers during training.

Note that the negative instances in the pretraining data are generated by shuffling the attribute values of the original data. So the the distributions of the attributes in the shuffled data are exactly the same as the original data, but the joint distributions of the attributes are different. By training the model to distinguish real data from the shuffled data, we hypothesize that the model has to learn the joint distributions between attributes. We also postulate that if the weights learned during pretraining are a better prior than commonly used initialization methods such as he_normal, then they should lead to better generalization.

5 Experiments

In this section, we present the proposed supervised pretraining experiments for binary classification. We evaluate the proposed method against he_normal initialization on UCI datasets in Section 5.1, and investigate when this method provides an increase in generalization using synthetic data in Section 5.2. In our experiments, we simulate a learning environment where labelled training data is limited but there is plenty of unlabelled data. The source code and data can be found on GitHub: github.com/superRookie007/supervised_pretraining.

5.1 Evaluation on UCI datasets

We conducted experiments on four UCI datasets to evaluate the proposed supervised pretraining against the he_normal [11] initialization method. The goal of

the experiments is to test if the proposed supervised pretraining leads to better test accuracy compared to the he_normal initialization. All the datasets were obtained from the UCI repository [5].

Table 2. Data information

Dataset	# Training examples	Class1	Class2	# Attributes
BanknoteID	288	55.5%	44.5%	4
BanknoteID_small	10	55.5%	44.5%	4
MAGIC	3995	64.8%	35.2%	10
MAGIC_small	10	64.8%	35.2%	10
Waveform	702	49.3%	50.7%	21
Waveform_small	10	49.3%	50.7%	21
Waveform_noise	695	50.0%	50.0%	40
Waveform_noise_small	10	50.0%	50.0%	40

Datasets BanknoteID is a dataset for identifying genuine banknotes from the forged banknotes. It has four continuous variables and 1372 data points in total. MAGIC is a simulated dataset for discovering primary gammas from the hadronic showers initiated by cosmic rays. It has 10 continuous variables and 19020 data points in total. Both Waveform and Waveform_noise are datasets for identifying different types of waves. Waveform has 20 attributes while Waveform_noise has 19 additional noisy variables drawn from a standard normal distribution. Both of them have 5000 examples in total. The original datasets of Waveform and Waveform_noise have three classes. We extracted two classes (class 1 and class 2) from the original datasets and treated them as binary classification problems. So the Waveform and Waveform_noise datasets in our experiments have 3343 and 3308 instances in total respectively.

Preprocessing Thirty percent (30%) of the data was used as the test set. The remaining data was split into labelled training data (21%) and unlabelled data (49%). Additionally, we created a smaller training set with only 10 instances for each of the four datasets by sampling the labelled training data. When sampling the datasets, we used stratified sampling. Table 2 shows some data characteristics of the training sets. When creating the pretraining data for our experiments, we combined the training data (ignoring the labels) and unlabelled data, and then applied the method described earlier in Section 4 to this combined data. This gave us an even bigger pretraining dataset. The pretraining data was scaled to a range [0, 1], then the same scaling parameters were applied to the test set.

Setup We used a five-layer fully connected network for all the experiments. Each layer has 1024 units except the last layer. The last layer is a softmax layer. The activation function used was ReLU. We used the standard full-batch gradient descent with a constant learning rate. Because the learning rate can potentially affect the generalization of neural networks, we used the same constant learning rate of 0.01 for all the experiments. We chose the number of epochs to make sure both models had converged. We consider the model has converged if the training loss stops decreasing and the training accuracy reaches 100%. In practice, a validation set is usually used to terminate the training early to avoid overfitting. However, we trained the models until convergence because we wanted a fair stopping criterion for both cases, and we wanted to eliminate the generalization effect and uncertainty of early stopping in the results. The number of epochs we used for each training set can be found in the second column of Table 3. Furthermore, the pretraining data was split into a training set (70%) and a validation set (30%). The pretraining stopped if the validation accuracy stopped increasing for the subsequent 100 epochs, and the model with the best validation accuracy was saved. We did not fine tune which layers to reuse or freeze the reused weights for a few epochs before updating them. We always transferred all the weights except the last layer and did not freeze any layer during training. The same experiment was run 10 times on each dataset. The experiments were all implemented using Keras [4] with the Tensorflow [1] backend.

Results The metric we used to measure generalization was accuracy on the test set. We have evaluated all the models with F1-score, AUC-ROC and Kappa coefficient, but they always moved in line with the test accuracy in this experiment. They did not add interesting information to the results, so they are not reported. The experimental results are shown in Table 3. The higher mean test accuracy is shown in bold, but the difference is not necessarily statistically significant. As expected, the test accuracy is generally higher when the training set is larger. And the standard deviation of the test accuracy tends to be larger when limited training data is available. The base models tend to have much lower training loss while the pretrained models tend to have lower test loss. When there is not a lack of labelled training data, there is no clear improvement on generalization using the supervised pretraining. On BanknoteID and Waveform_noise, the pretraining actually hurt the generalization. However, when the training set is very small, only 10 instances in this case, the mean test accuracy for BanknoteID and Waveform_noise of the pretrained model became higher than that of the base model. It is interesting that the improvement in generalization by using the pretraining was big on Waveform_noise_small, but this was not the case on Waveform_small. Recall the difference between Waveform_noise and Waveform is that Waveform_noise has 19 additional noisy features drawn from the standard normal distribution. It is not very clear if the difference in the results was due to the increased dimensionality or the characteristics of the additional noisy features. We conducted further experiments on synthetic data in order to in-

Table 3. Comparison of the model initialized using the supervised pretraining method (Pretrained) against the model initialized with he_normal (Base). We ran each case 10 times. The training accuracy reached 100% for all the runs.

Data	Epochs	Method	Train loss	Test loss	Test acc.
BanknoteID	10000	Base	9.78E-04 ±2.57E-05	5.79E-02 ±1.38E-03	**99.32%** ±0.28%
		Pretrained	1.04E-02 ±2.99E-04	4.30E-02 ±5.53E-03	98.74% ±0.39%
BanknoteID_small	5000	Base	1.93E-04 ±5.40E-06	9.02E-01 ±3.16E-02	82.18% ±0.80%
		Pretrained	1.96E-02 ±7.88E-04	6.13E-01 ±2.67E-02	**82.45%** ±0.81%
MAGIC	100000	Base	4.15E-03 ±4.88E-04	9.15E-01 ±1.71E-02	85.42% ±0.23%
		Pretrained	8.59E-03 ±3.21E-03	7.12E-01 ±2.54E-02	**85.69%** ±0.27%
MAGIC_small	5000	Base	2.37E-04 ±3.93E-06	1.22E+00 ±5.87E-02	78.12% ±0.20%
		Pretrained	2.04E-02 ±7.59E-04	7.20E-01 ±3.54E-02	**78.62%** ±0.51%
Waveform	20000	Base	5.71E-04 ±3.82E-05	3.02E-01 ±6.99E-03	94.10% ±0.25%
		Pretrained	9.89E-03 ±6.39E-03	2.36E-01 ±1.24E-02	**94.38%** ±0.25%
Waveform_small	10000	Base	8.97E-02 ±2.83E-01	4.06E-01 ±3.02E-02	88.48% ±0.58%
		Pretrained	1.02E-02 ±2.20E-04	4.13E-01 ±2.96E-02	**89.10%** ±0.67%
Waveform_noise	20000	Base	3.13E-04 ±1.70E-05	3.15E-01 ±9.78E-03	**94.72%** ±0.30%
		Pretrained	1.38E-02 ±6.81E-03	2.23E-01 ±9.31E-03	94.54% ±0.34%
Waveform_noise_small	10000	Base	4.85E-05 ±1.35E-06	6.33E-01 ±4.43E-02	82.38% ±0.57%
		Pretrained	1.01E-02 ±4.17E-04	5.03E-01 ±1.58E-01	**86.97%** ±3.33%

vestigate when the supervised pretraining helps improve generalization. These experiments are presented in the next section.

5.2 When to use the supervised pretraining

The experiments discussed here investigate the circumstances, where the proposed pretraining holds an advantage over the he_normal initialization in terms of test accuracy. More specifically, we test the effect of data dimensionality, different types of noisy features, size of labelled training data and distance between class clusters on the performance of the proposed pretraining. All the experiments were conducted on synthetic datasets generated using the *make_classification* API in *sklearn* library. This is an adapted implementation of the algorithm used to generate the "Madelon" dataset [10].

Firstly, we test how data dimensionality, noisy features and redundant features affect the effectiveness of the supervised pretraining compared to he_normal. We generated 6 different datasets. All of the datasets have only two classes and have balanced class distribution. Each of the raw datasets had 10000 instances. The *class_sep* parameter was set to 0.01 for all datasets (the smaller the *class_sep*, the more difficult the classification). We applied the same preprocessing as the previous experiment. But in this experiment, we sampled a training set with 50 instances for each dataset. All the other experiment setups were exactly the same

as described in Section 5.1. The characteristics of the 6 datasets are described below.

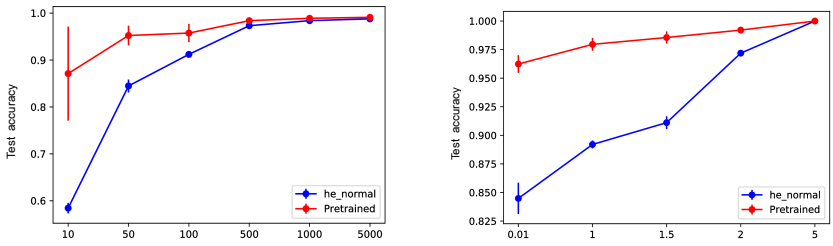
- Informative_10 has 10 attributes and all of the attributes are informative for the classification task.
- Informative_20 has 20 attributes and all of the them are informative.
- Redundant_20 has 20 attributes, but 10 of them are the exact copy of the other 10 informative attributes.
- In Std_normal_20, 10 of the 20 attributes are random values drawn from the standard normal distribution (mean 0, standard deviation 1), while the other 10 are informative.
- In Normal_20, 10 of the attributes are drawn from normal distributions whose means and standard deviations are kept the same as the other 10 informative attributes.
- In Shuffled_20, instead of drawn randomly from normal distributions, the 10 noisy attributes are simply shuffled informative attributes (the distribution of each of the noisy attributes is the same as the corresponding informative attribute).

Table 4. Investigating the effect of dimensionality, noisy and redundant features on the performance of the supervised pretraining. We ran each case 10 times and all of the models were trained for 10000 epochs.

Data	Method	Train loss	Train acc.	Test loss	Test acc.
Informative_10	Base	6.50E-04 ±2.46E-05	100.00%	5.79E-01 ±6.89E-02	84.48% ±1.37%
	Pretrained	1.02E-02 ±1.41E-04	100.00%	1.43E-01 ±2.69E-02	**96.24%** ±0.77%
Informative_20	Base	3.56E-04 ±1.76E-05	100.00%	1.82E+00 ±7.84E-02	63.61% ±0.72%
	Pretrained	1.03E-02 ±2.87E-04	100.00%	4.71E-01 ±1.02E-01	**87.19%** ±2.64%
Redundant_20	Base	6.07E-04 ±2.50E-05	100.00%	5.86E-01 ±4.82E-02	84.41% ±0.93%
	Pretrained[1]	1.10E-02 ±7.33E-04	100.00%	4.64E-01 ±4.32E-02	**85.48%** ±1.35%
Std_normal_20	Base	3.87E-04 ±2.01E-05	100.00%	2.40E+00 ±7.50E-02	59.94% ±0.64%
	Pretrained	1.02E-02 ±2.83E-04	100.00%	2.13E-01 ±3.53E-02	**94.18%** ±1.09%
Normal_20	Base	4.12E-04 ±1.43E-05	100.00%	1.24E+00 ±5.77E-02	65.40% ±0.92%
	Pretrained	1.02E-02 ±1.60E-04	100.00%	1.84E-01 ±2.92E-02	**94.93%** ±0.87%
Shuffled_20	Base	3.42E-04 ±1.35E-05	100.00%	2.42E+00 ±8.56E-02	57.95% ±0.39%
	Pretrained	1.02E-02 ±1.15E-04	100.00%	1.74E-01 ±2.34E-02	**95.19%** ±0.55%

The results are shown in Table 4. The base models for Informative_10 and Redundant_20 performed very similarly. This is not too surprising considering

[1] When running the pretrained model of Redundant_20, 4 out of 10 runs failed to converge, the results shown here were collected from the 6 converged runs.



(a) Size of training set (b) Class separation

Fig. 4. Comparison of the model trained with supervised pretraining (red) against the model initialized with he_normal (blue) in terms of test accuracy. The plots show the distributions of test accuracy. Each experiment was run 10 times.

Redundant_20 basically concatenated two copies of the features in Informative_10 together. Compared to Informative_10, Redundant_20 neither has any additional information nor lacks any useful information. The improvement on generalization using the supervised pretraining was around 11% on Informative_10. But for Redundant_20, the pretrained model failed to converge in 4 out of 10 runs. And there is no obvious improvement on test accuracy when pretraining is used. The generalization gain on Informative_20 was more than 20%, while the gain on the Std_normal_20 was around 34%. The biggest gain (around 37%) occurred in Shuffled_20 dataset. The results suggest that both the increased dimensionality and the noisiness of the additional features both contribute to the generalization improvement we observed on Waveform_noise_small in the previous experiment.

Next, we investigated how the size of labelled training data and the class separation would affect the generalization advantage of the proposed pretraining over the he_normal initialization. The experimental setup was exactly the same as the previous experiments, except the datasets used. When testing the effect of the size of labelled training data, we held all other factors constant including the class separation ($class\_sep = 0.01$) and unlabelled data. The data had 10 attributes and all of them were informative. We tested six labelled training sets with 10, 50, 100, 500, 1000 and 5000 instances respectively. We ran each experiment 10 times. The results can be found in Figure 4(a). The results show that as the size of labelled training data increases, the advantage of the pretraining gets smaller. And as expected, as the training set gets larger, the standard deviation of test accuracy shrinks. Finally, we tested the effect of class separation. Class separation in this case means the distance between classes. The smaller the class separation, the harder the classification. We created three datasets with three different levels of class separation by setting the *class_sep* parameter to 0.01, 1, 1.5, 2 and 5 respectively. All of the datasets had only 10 informative attributes. The labelled training sets had 50 instances for these experiments. Again, all the other experiment setups were the same as described earlier. Figure 4(b) shows the results of these experiments. The pretraining had the biggest gain in test

accuracy over he_normal when the class separation is 0.01. However, as the class separation gets larger the advantage becomes smaller. Both of these experiments support the hypothesis that what we are learning during pretraining is a good prior for the model. The advantage of a good prior is larger when the data size is small and when the classification problem is difficult.

We proposed a new supervised pretraining method to help improve generalization for binary classification problems. The experiments on both UCI and synthetic datasets suggest that the proposed pretraining method provides more generalization gain when there is a lack of labelled training data. Further experiments on synthetic data suggest that this method works best when the input has high dimensionality and contains noisy attributes. Finally, the proposed method has bigger advantage over he_normal in terms of test accuracy when the class separation is small.

6 Discussion

By inspecting the results closely, we noticed that the standard deviation of the test accuracy for the pretrained model was bigger than the base model without pretraining for a few cases. This is contradictory to the findings in unsupervised pretraining. Unsupervised pretraining was shown to help lower the variance of test accuracy [6]. And the results in Table 3 are not statistically significant. One possible explanation for this might be the weights learned during the pretraining across different runs were very different. Recall that we used a pretraining validation set to stop the pretraining. Another way to do this is to run the pretraining using different epochs and choose the epoch that gives the best validation accuracy on the original classification problem. Then we can run the experiments multiple times using this fixed number of epochs. This may be a more stable method. Another possible explanation for the large standard deviation is the small size of the training data. The standard deviation of the test accuracy tends to be larger when training data is small. The difficulty of the classification problem may also affect the statistical significance of the results. As the experiments on synthetic data indicate, the advantage of the proposed pretraining method disappears when the clusters are far apart from each other. The complexity of the target decision boundary that separates different classes may also affect the effectiveness of the pretraining method. Lastly, we also expect label noise in the data to reduce the advantage provided by the pretraining. The exact reasons why the proposed method works better on certain datasets need to be explored further.

The pretraining phase means more training time. However, the proposed supervised pretraining only adds a small amount of additional training time. For instance, the pretraining phase for the MAGIC dataset took on average 241 seconds, about 4.67% of the average total training time of 5061 seconds. Note that the exact training time in practice depends on the dataset, tuning of training parameters and the number of epochs used. We just want to show that despite of the additional pretraining phase the proposed method is still practical.

Note that in all of our experiments, we did not fine tune the pretraining or the weight transferring process. One can possibly improve the results further by carefully and tediously choosing when to stop the pretraining, which layers to reuse and whether to freeze some layers for a certain number of epochs. Our goal here is mainly to show that the proposed method can help improve generalization when only limited training data is available, instead of achieving state-of-the-art result on a particular dataset. We used the same experimental setup across all the experiments (except number of epochs) to not bias any particular setting.

7 Conclusion and future works

Current default initialization methods such as he_normal are designed to stabilize the training, especially for deep neural networks. However, they are not necessarily optimal for generalization. When labelled data is limited, the choice of initial weights becomes very important in deciding what final model we will end up with. We demonstrated that inappropriate initial weights of neural networks do indeed lead to lower test accuracy. We then proposed a supervised pretraining method to improve the generalization in binary classification problems. During the pretraining, a model is learned to identify real data from shuffled data. Then the learned weights are reused in the original problem. Based on the experimental results on four UCI datasets and synthetic datasets, the supervised pretraining leads to better test accuracy than he_normal initialization, when there is a lack of labelled training data and the class separation is small. The experiments using synthetic datasets showed that this supervised pretraining works best on datasets with higher dimensionality and noisy features.

A very important future work is to understand why this supervised pretraining works in some of the cases and why it fails in other situations. We only focused on binary classification problems in this work, we would like to evaluate how well the proposed method works on multi-class problems. Lastly, it would be interesting to explore how to apply the same idea to image and text data.

References

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/, software available from tensorflow.org
2. Baldi, P., Sadowski, P., Whiteson, D.: Searching for exotic particles in high-energy physics with deep learning. Nature communications **5**, 4308 (2014)
3. Bengio, Y., Lamblin, P., Popovici, D., Larochelle, H.: Greedy layer-wise training of deep networks. In: Advances in neural information processing systems. pp. 153–160 (2007)

4. Chollet, F., et al.: Keras. https://github.com/fchollet/keras (2015)
5. Dheeru, D., Karra Taniskidou, E.: UCI machine learning repository (2017), http://archive.ics.uci.edu/ml
6. Erhan, D., Bengio, Y., Courville, A., Manzagol, P.A., Vincent, P., Bengio, S.: Why does unsupervised pre-training help deep learning? Journal of Machine Learning Research **11**(Feb), 625–660 (2010)
7. Glorot, X., Bengio, Y.: Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics. pp. 249–256 (2010)
8. Graves, A., Mohamed, A.r., Hinton, G.: Speech recognition with deep recurrent neural networks. In: 2013 IEEE International Conference on Acoustics, Speech and Signal Processing. pp. 6645–6649. IEEE (2013)
9. Gulshan, V., Peng, L., Coram, M., Stumpe, M.C., Wu, D., Narayanaswamy, A., Venugopalan, S., Widner, K., Madams, T., Cuadros, J., et al.: Development and validation of a deep learning algorithm for detection of diabetic retinopathy in retinal fundus photographs. Jama **316**(22), 2402–2410 (2016)
10. Guyon, I.: Design of experiments of the NIPS 2003 variable selection benchmark (2003)
11. He, K., Zhang, X., Ren, S., Sun, J.: Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Proceedings of the IEEE international conference on computer vision. pp. 1026–1034 (2015)
12. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
13. Hinton, G.E., Osindero, S., Teh, Y.W.: A fast learning algorithm for deep belief nets. Neural computation **18**(7), 1527–1554 (2006)
14. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
15. Mishkin, D., Matas, J.: All you need is a good init. In: Proceedings of the International Conference on Learning Representations (ICLR) (2016)
16. Ranzato, M., Poultney, C., Chopra, S., Cun, Y.L.: Efficient learning of sparse representations with an energy-based model. In: Advances in neural information processing systems. pp. 1137–1144 (2007)
17. Sak, H., Senior, A., Rao, K., Beaufays, F.: Fast and accurate recurrent neural network acoustic models for speech recognition. In: Sixteenth Annual Conference of the International Speech Communication Association (2015)
18. Saxe, A.M., McClelland, J.L., Ganguli, S.: Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. arXiv preprint arXiv:1312.6120 (2013)
19. Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Advances in neural information processing systems. pp. 3104–3112 (2014)
20. Yosinski, J., Clune, J., Bengio, Y., Lipson, H.: How transferable are features in deep neural networks? In: Advances in neural information processing systems. pp. 3320–3328 (2014)